编译原理实验四报告

191870271 赵超懿

实现功能

1. 完成了将中间代码转化未MIPS32汇编代码
2. 汇编代码支持多维数组和结构体，同样支持这些数据结构的参数传递
3. 寄存器的使用基本遵循了手册的约定

程序使用及编译

1. 在Code文件夹下运行make parser生成二进制文件parser
2. 运行命令./paser $FILE $FILE即可
3. make test做了一些修改，详细情况见makefile

代码实现

寄存器分配

采用局部寄存器分配算法（讲义中的方法），同时计算中的寄存器仅使用$t0-t7, s1-s7$, 剩下的寄存器中$v0$寄存器作为存储返回值的寄存器，$v1$作为临时赋值的寄存器，$a0-a3$作为参数传递寄存器， 同时$s8$寄存器用作fp进行栈的管理

栈结构设计

在我的设计中，栈存放所有的临时变量。函数调用采用手册要求，$a0-a3$用于参数传递，剩余变量放置于栈上。fp作为栈底，sp作为栈顶

```
|  args |    多于4个参数放在栈上
|  $fp  |    保存的当前函数的栈底
|  $ra  |    返回的PC值
| func  |    进入被调用的函数
```

具体实现流程

1. 将中间代码分成基本块
2. 从后到前扫描基本块，获得每条中间代码的变量的在各个程序点的使用信息
3. 从前到后扫描一次，为变量分配在栈中的存储位置
4. 以基本块为单位生成MIPS汇编代码

生成过程中溢出变量的处理使用讲义中的方法，将寄存器变量写回，同时在调用函数前也写回以保存现场。

在我的中间代码中，除了直接声明的变量对应的中间代码变量外，其他中间代码变量都为单赋值，所以在生成每个基本块的汇编代码后，在将寄存器写回内存中时，可以根据寄存器中存储的变量类型来确定是否需要写回，减少了写回的次数。

解决的问题

1. 何时保存当前寄存器中的值

   首先，对于每个基本块结束后，应该进行一次保存。其中对于以if goto, return , goto为最后指令的基本块（即可能发生跳转的指令），应当在最后一条指令前之前进行保存；此外，对于函数调用前，也需要进行寄存器的保存。

   在函数调用的寄存器保存中，没有遵循手册中的要求，我的实现为全部由调用者进行保存。

2. 函数调用的活动记录以及变量访问

   我采用讲义中使用fp寄存器的实现，fp寄存器在每个函数中是固定的，确定好变量相对fp的位置后，便可以直接访问，对应的sp寄存器可能随时变化。活动记录见上文的栈结构设计。

3. 代码设计

   参考了讲义的建议，和中间代码生成的gencode对应，使用一个emit_code函数，负责生成代码。

   第四次实验的代码比较短，相较于前几次实验明显要少很多，所以写起来比较好写。

存在的问题

没有做什么优化，所以生成的汇编代码比较冗余，使用的空间很多